Introduction to Computer Organization – Fall 2023

Lab 3

Due: @11:55 PM, Wed September 20th via Gradescope

The following assignment is intended to be completed during your assigned lab period. One member of your group must submit the assignment to Gradescope by the posted deadline and indicate your group members when submitting the assignment. **Each group member must be present during the scheduled lab period in order to receive credit.**

Group names and uniqnames

Pod Member Name	**Uniqname**
Yuzhen Chen	yuzhench
Yuhan Zhang	zyuhan

For each of the following problems, one person should act as the "scribe" and log the discussions of the group. You should rotate who is the scribe for each problem and indicate in the given space.

Problem 1: GodBolt *[20 Points]*

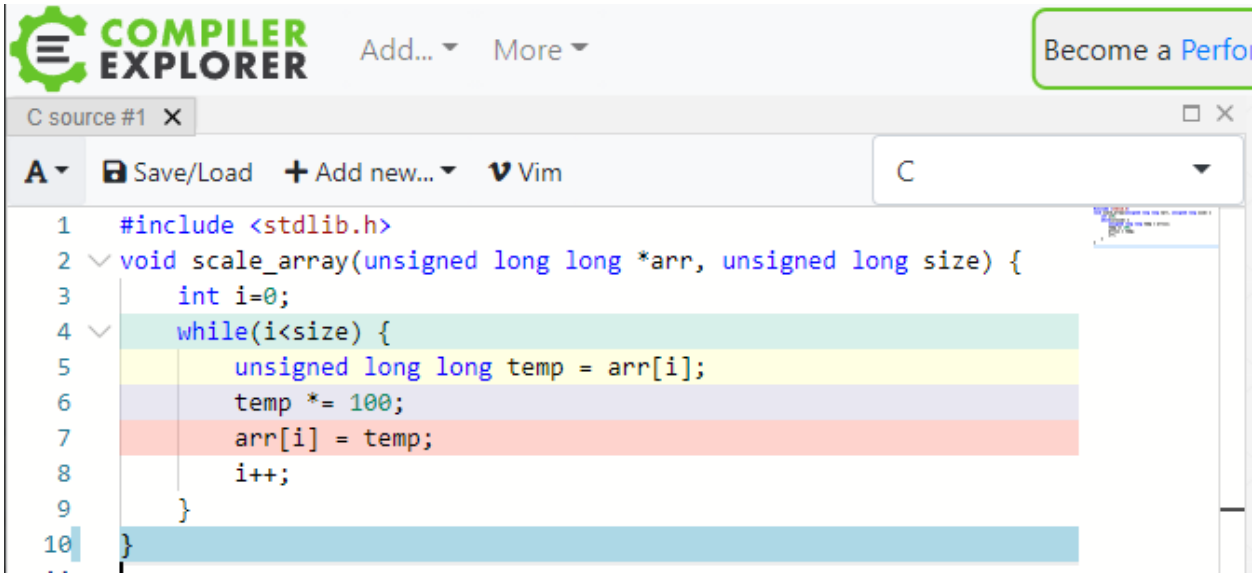
Scribe: **[Scribe's name here]**

Go to godbolt.org. This is a website that will compile source code from several different programming languages into one of many target assembly languages, and color code the correspondence between the two programs.

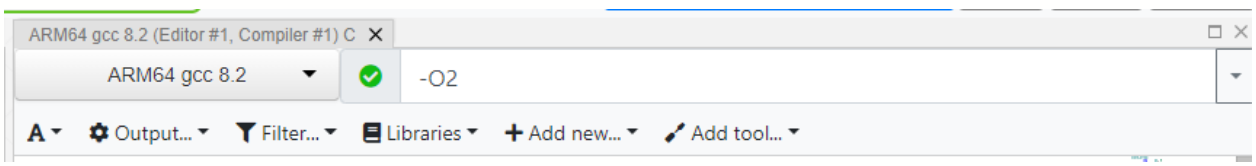
In the left hand window, select "C" is the source language and paste in the following code:

```
#include <stdlib.h>
void scale_array(unsigned long long *arr, unsigned long size) {
    int i=0;
    while(i<size) {
        unsigned long long temp = arr[i];
        temp *= 100;
        arr[i] = temp;
        i++;
    }
}
```

Your window should look like this (with perhaps rows highlighted as different colors)



In the right hand window,select "carm64g820" or "ARM64 gcc 8.2" for the architecture and "-O2" for the compiler options:



After a moment, the right window should be populated with ARM assembly code. You can see which C statements each instruction corresponds to by matching the colors between the two windows.

Because this compiler is using the full Arm assembly language (not just the LEG subset described in lecture), there are a few instruction variants you may be unfamiliar with (note that we do **NOT** expect you to be familiar with these instructions outside the scope of this problem). The first is:

```
add x1, x2, x3, lsl 4
```

This instruction combines an `add` instruction and an `lsl` instruction, which could be translated to:

```
x1 = x2 + (x3 << 4)
```

The second new instruction looks like:

```
str x0, [x1], 4
```

This instruction simply combines a store instruction with an `add` instruction. It is equivalent to:

```
sturw x0, [x1,#0]
add x1, x1, #4
```

1. In the space, below paste the compiled assembly and explain how each instruction is implementing the specified C code. In particular, describe how the assembly multiplies an element by 100 without the use of an explicit multiply instruction. **[10 pts]**

Change the source code above so that rather than multiplying each element of the array by 100, it instead multiplies each element by 127.

2. Briefly (30 words or fewer) describe the change in the resulting assembly and how it accomplishes the multiplication. **[10 pts]**

$2^0 = 1$ $2^1 = 2$ $2^2 = 4$ $2^3 = 8$

2 128 128.

$2^6 = 64$ $2^7 = 128$

2^8.

Problem 2: Struct Addresses *[10 points]*

Scribe: **[Scribe's name here]**

List an ordering of the variables below that minimizes total memory used. Determine the start and end addresses for the following variables (decimal addresses are OK). *(Assume a 64-bit system and that the data starts on address 200_{10})*

```
struct {
  short a;
  int b;
  char c;
  int* d;
  struct {
      int e;
      char f [10];
  } g;
} example;
```

Variable	Start Address	End Address

Problem 3: Putting it All Together *[20 points, 1 pt per line]*
Scribe: **[Scribe's name here]**
Note: This is a past exam problem. This is hard, but representative of exam questions. We've filled in some of the blanks to make it easier for lab.

Convert the following C code to ARM. Assume the system requires memory alignment, and that the assembler will not fix opcodes. Use labels when possible. The return value should be in register *X2*. Assume:

- Starting address of the **data** array is stored in register ***X0***, and consists of **512 elements.**
- Assume all operations will be within the **32 bit unsigned integer limits.**

Code explanation (you don't need to read this to answer the question):
The ABC370 factory is responsible for producing packs of 20 pencils. However, this factory is not always perfect, which means some packs will contain more or less than the desired amount of 20 pencils. In the case where there are over 20 pencils per pack, we want to integer divide the count by 2. In the case where there are less than 20 pencils per pack, we want to first figure out if the count is an even number. If it is an even number, we simply multiply by 4, otherwise we would add 8. Return the number of pencil packs not changed in register X2.

C	ARM

`struct pencil_pack{`
`  char pack_id[20]; //20B`
`  int64_t num_penc; //8B`
`};`
 	
`int32_t`
`check_pencil_arr(`
`  struct pencil_pack data[]) {`
`  int32_t good_count = 0;`
 	
`  for(int32_t i = 0; i < 512; ++i){`
`  if(data[i].num_penc < 20){`
`  if(data[i].num_penc % 2 == 0){`
`  data[i].num_penc *= 4;`
`  } else {`
`  data[i].num_penc += 8;`
`  }`
`  } else if(data[i].num_penc > 20){`
`  data[i].num_penc /= 2;`
`  } else {`
`  ++good_count;`
`  }`
`  }`
 	
`  return good_count;`
`}`
	`func:`
	`  MOVI X2, #0x00`
	`  MOVI X3, #0x00`
	`loop:`
	`  CMPI X3, #512`
	`  B.EQ end`
	`  LSL X4, X3, #3`
	`  ADD X4, X4, X0`
	`  LDUR X5, [X4, #20]`
	`  CMPI X5, #20`
	`  B.GT bigger`
	`  B.LT smaller`
	`  ADDI X2, X2, #1`
	`  B forend`
	`bigger:`
	`  LSR X5, X5, #1`
	`  STUR X5, [X4, #24]`
	`  B forend`
	`smaller:`
	`  ANDI X7, X5, #1`
	`  CMPI X7, #0`
	`  B.EQ even`
	`  ADDI X5, X5, #8`
	`  STUR X5, [X4, #24]`
	`  B forend`
	`even:`
	`  LSL X5, X5, #2`
	`  STUR X5, [X4, #24]`
	`forend:`
	`  ADDI X3, X3, #1`
	`  B loop`
	`end:`